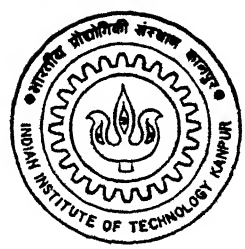
A CORPUS-BASED APPROACH FOR GENERATING VIBHAKTI-CHARTS IN INDIAN LANGUAGES

by

SARANGA DHAR SAMANTARAY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST, 1995

A CORPUS-BASED APPROACH FOR GENERATING VIBHAKTI-CHARTS IN INDIAN LANGUAGES

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

By

Saranga Dhar Samantaray

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING.

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR.

AUGUST, 1995

CSE-1995-M-SAM-COR

Certificate

It is certified that the work contained in the thesis entitled **A CORPUS-BASED APPROACH FOR GENERATING VIBHAKTI-CHARTS IN INDIAN LANGUAGES** , by *Saranga Dhar Samantaray* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Rajeev Sangal

(Dr Rajeev Sangal)
Professor
Department of Computer Science & Engineering
Indian Institute of Technology
Kanpur–208016 India

August, 1995.

ABSTRACT

A corpus-based approach is used to obtain vibhakti-chart (or subcategorization frames) from a corpus of Kannada using a Kannada to Hindi anusaaraka.

Anusaaraka is used to perform morphological analysis, as well as to produce anusaaraka Hindi output of the given corpus. The morphological analyzer's output is filtered to handle participle cases, avyaya anomaly cases, kriya-mula cases, relational word cases, samaasa cases, ellipsis & anaphora cases, and sambhodhan etc., cases. Occurences of different vibhaktis of nouns in simple sentences for a given verb *baru (come)* are counted, which shows the relative occurence frequency of different vibhakti with this verb. Then vibhakti-vectors are generated. Vector average is taken using a weighted vector distance function to obtain the vibhakti-chart for the verb.

This is tested on a part of the corpus not used above. The approach shows promise in assisting for the construction of vibhakti-charts, which are a part of karaka-charts. These in turn are important in karaka disambiguation.

Acknowledgements

It is a great pleasure to place on record my deep sense of gratitude to my thesis supervisor *Dr. Rajeev Sangal*, for his guidance and support throughout this work. No words are adequate to quantify his ever smiling face, patience, tolerance and understanding. His constant supervision and guidance have been exemplary and worth emulating. Without his support, this thesis would not have materialized.

It has been a great pleasure to learn in association with *Dr. Sanjeev Kumar*, who in fact introduced IIT,K to me, and remain in constant touch with me, through out my stay at IIT,K in either one or other capacity. I wish to thank him for his concern, affection and encouragement.

I wish to thank Akshar group, in general and Mrs. Amba Kulkarni, Mrs. Gangadharia, and Mr. Jha, in particular for their help in understanding the points.

The work reported in this thesis has been carried out with the help of Kannada anusaaraka and the Kannada corpus available at IIT,K. This support is gratefully acknowledged.

I gratefully acknowledge the help and encouragement given by all the faculty and staff members of CSE department in particular and of IIT,K in general. I also thank the office staff, library staff and the lab. staff of CSE, who were always ready to provide all possible help.

I express my deep sense of gratitude to my beloved, for constant encouragement, moral support and care. All my well wishers were of great help to me, when I was passing through a difficult and unimaginable situation. My humble respect and gratefulness to each one of them.

I take this oppurtunity to thank all inmates of Hall-5, particularly of G-top, who made my stay at IIT,K a memorable one. It was really a nice experience to cherish the company of Kushal,Patnaik,Dobal and Anand.

29th August, 1995 **S.D.Samantaray**

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Resolving syntactic ambiguities is still one of the biggest problem in machine translation systems. The goal of designing a computer system, capable of conversing with people in their own natural language has been a dream of A.I., almost since the inception of digital computers in the 1940's. To be able to converse in human language is a basic pre-requisite of any intelligent assistant, because language serves as our basic vehicle for thought and communication.

Natural languages are communication tools which constantly evolve and therefore are highly complex in nature. The presence of ambiguities at all levels is the source of complexity of natural languages, and it constitutes a major problem for automatic language processing.

Producing an unambiguous parse is a major challenge for the parsers developed for Indian Languages. Correct verb and noun group attachment poses the greatest hindrance in this regard. However, Paninian parser has made some attempts to come up with a solution for selecting the correct parse out of multiple parses. It has incorporated parsing based on integer programming, graph matching and assignment [5].

1.2 Basis of the work

With the easy availability of electronic corpus, nowadays people have started adopting corpus-based approaches for solving various machine translation problems. The corpus-

based approach has been found to be very effective in resolving ambiguous prepositional phrase (PP) attachments in the English language [8, 12, 13].

Ambiguous PP attachments are resolved on the basis of lexical association derived from distribution of lexical items in the text corpus [12], and also on the basis of subcategorization information extracted for a verb from the corpus [8, 13]. A subcategorization frame (subcat. frame) is a statement of what types of syntactic arguments a verb takes.

The verb and noun group attachment problem in Indian Languages is also a case of deciding correct attachment, somewhat similar to the PP attachment problem of English.

Further, in reference to Paninian parser a karaka-chart expressing restrictions of demand-group on source-group is similar to subcategorization and selectional restriction [5].

On the basis of the above points we feel that development of a chart for a verb (similar to subcat. frame) expressing the restrictions of attachments for nouns with that particular verb, could be very effective in defining correct verb-noun attachments. we call it *vibhakti chart* because vibhakti is the basic unit of consideration.

We illustrate our governing idea in the following way :

- In Indian languages, vibhaktis are different for attaching to noun and verb. Therefore vibhaktis very clearly indicate what nouns are related to what verbs.
- Using the corpus-based approach, we can extract actual vibhaktis with nouns for a given verb in sentences, and can store the vibhakti information (list of vibhaktis) so obtained. If using this we create a vibhakti chart (or subcat frame), then this vibhakti chart can play a pivotal role in defining better restrictions, and can be used more effectively for karaka disambiguation.

1.2.1 Problem of karaka disambiguation :

The problem of karaka disambiguation could be better understood if we go through the details, involved in the process of parsing. (This section is taken from [4]). The steps of parser have been shown in the figure 1.1.

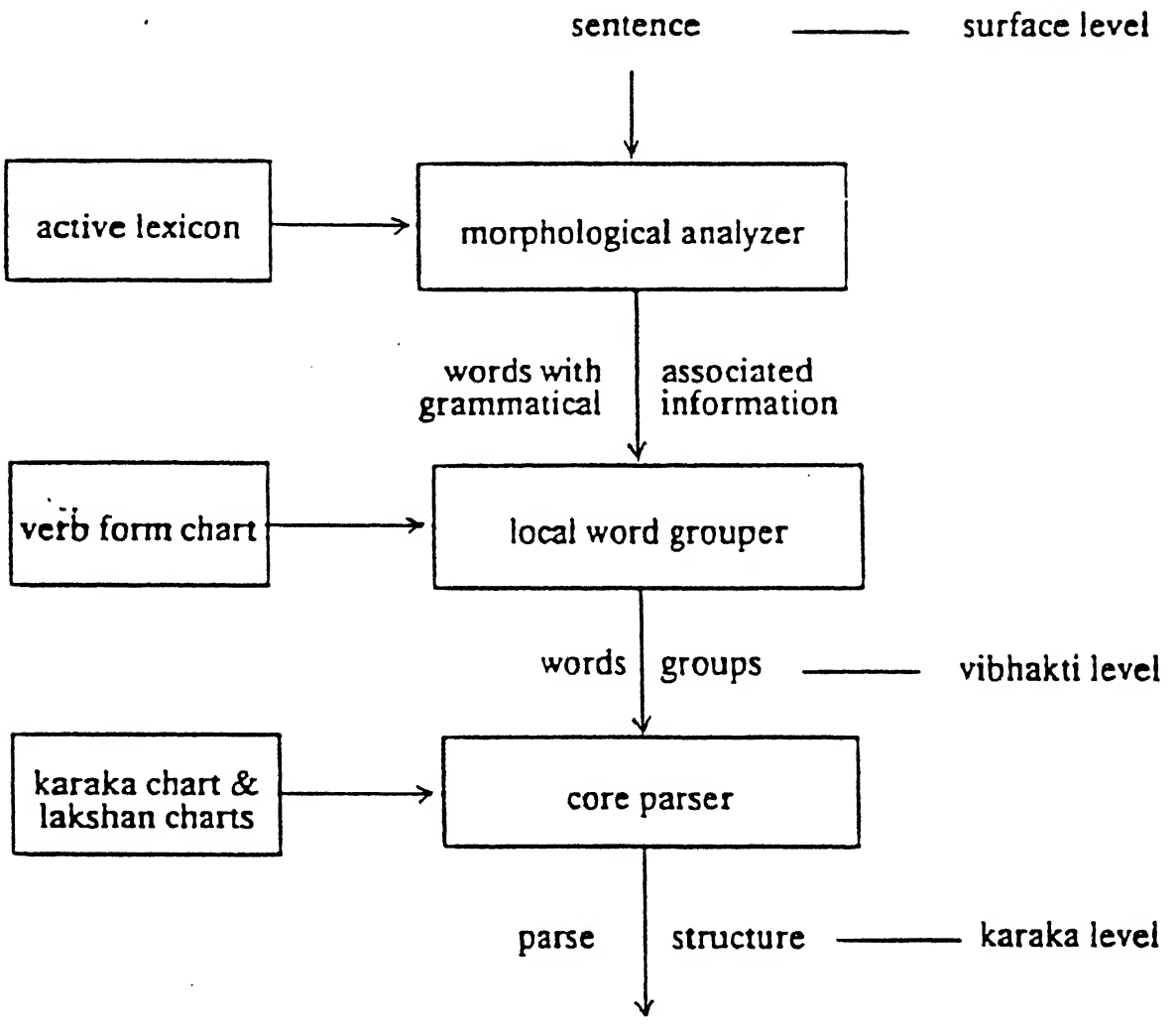


Figure 1.1: Structure of the Parser

After the local word grouping stage ([4],Chap4), there is karaka assignment and lexical disambiguation stage. This is done by the core parser.

Given the local word groups in a sentence, the task of the core parser is two-fold:

1. To identify karaka relations among word groups, and

2. To identify senses of words.

The first task requires knowledge of karaka-vibhakti mapping, optionality of karakas, and transformation rules. The second task requires lakshan charts for nouns and verbs.

A data structure called *karaka chart* stores information about karaka-vibhakti mapping and optionality of karakas for each of the verb groups in a sentence. Initially, the default karaka chart is loaded into a karaka chart for a given verb group in the sentence. Transformation is performed using the TAM label. There is a separate karaka chart for each verb group in the sentence being processed.

An example default karaka chart for KA (eat) has been shown. It shows for each of the karakas its necessity (mandatory, desirable, or optional), and vibhakti. These are called *karaka restrictions* in a karaka chart.

KARAKA	VIBHAKTI	PRESENCE
Karta	ϕ	mandatory
Karma	ko or ϕ	mandatory
Karana	se or dvArA	optional

Figure 1.2: Default karaka chart for KA

For a given sentence after the word groups have been formed, karaka charts for the verb groups are created and each of the noun groups is tested against the karaka restrictions in each karaka chart (provided the noun group is to the left of the verb group whose karaka chart is being tested). When testing a noun group against a karaka restriction of a verb group, vibhakti information is checked, and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group. This can be shown in the form of a constraint graph. Nodes of the graph are the word groups and there is an arc from a verb group to a noun group labeled by a karaka, if the noun group satisfies the karaka restriction in the karaka chart of the verb group. (There is an arc from one verb group to another, if the

karaka chart of the former has a karaka restriction with lexical type as verb or sentence.) The verb groups are called demand groups as they make demands about their karakas, and the noun groups are called source groups because they satisfy demands. (A verb group can be a source group as well when it satisfies the demand of another verb group. This, however, does not affect its status as a demand group as well.)

As an example, let's consider a sentence containing the verb KA (eat):

```
S.1  baccA  hATa se   kelA   KAtA hE.
     child hand -se  banana  eats
     (The child eats the banana with his hand.)
```

Its word groups are marked and KA (eat) has the same karaka chart as in fig.1.2. Its constraint graph is shown in figure 1.3.

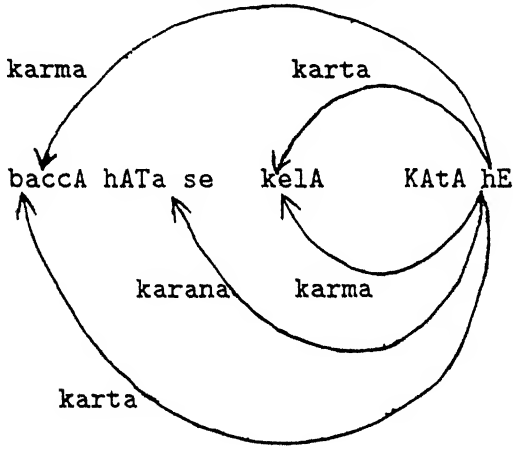


Figure 1.3: Constraint graph for sentence S.1

1.2.2 Constraints

A parse is a sub-graph of the constraint graph satisfying the following conditions:

C1. For each of the mandatory karakas in a karaka chart for each demand group, there should be exactly one outgoing edge labeled by the karaka from the demand group.

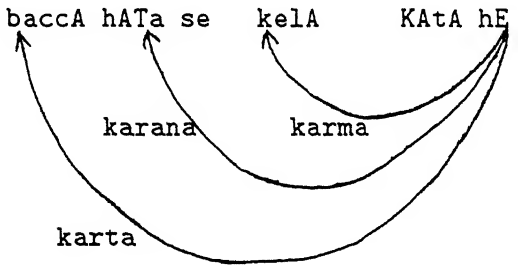
C2. For each of the optional karakas in a karaka chart for each demand group, there should be at most one outgoing edge labeled by the karaka from the demand group.

C3. There should be exactly one incoming arc into each source group.

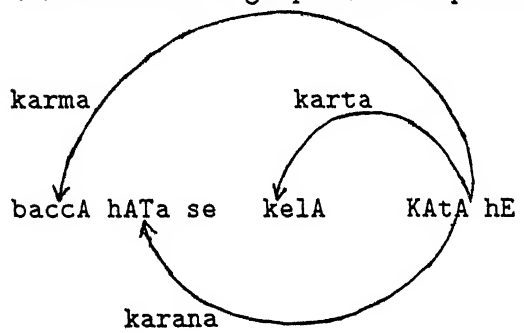
1.2.3 The problem defined

There may be situations where multiple sub-graphs of a constraint graph might be satisfying the constraints. If several sub-graphs of a constraint graph satisfy the above conditions, it means that there are multiple parses and the sentence is ambiguous. In other words, there exists at least one source group for which proper karaka assignment can not be adjudged correctly. Thus there exists an ambiguity in *karaka assignment.*

The example constraint graph of figure 1.3, suffers from ambiguity in karaka assignment as it can have two possible parses. Figure 1.4 shows the two possible parses for the constraint graph.



(a) A solution graph (corresponding to the meaning: child eats banana)



(b) Another solution graph (meaning: banana eats child)

Figure 1.4: Solution graphs for sentence S.1

Resolving such situations, *i.e.,* producing an unambiguous parse, in general, can be termed as *karaka disambiguation.*

1.3 Conventional karaka disambiguation methodologies

Following methodologies have been identified for producing an unambiguous parse when, for a given verb, karaka-vibhakti mapping is not sufficient.

Using World Knowledge : World knowledge can be used during parsing, to apply preferences over parses. But such knowledge explodes in size very fast, and is very difficult to use during parsing (or processing).

Including Semantic Types : It suggests to include semantic types under ambiguity conditions. The semantic types so included have the sole-purpose of karaka disambiguation. This keeps the number of semantic types under control, and serves as a guiding philosophy for what semantic types to include. Fig. 1.5 shows the starting semantic type hierarchy which is sufficient for a major part of language.

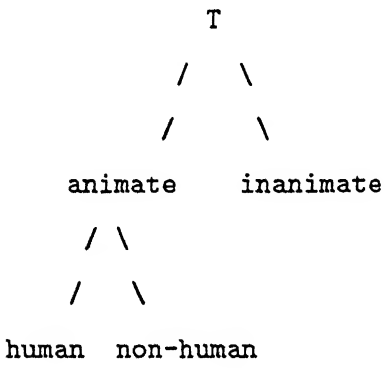


Figure 1.5: Semantic type hierarchy

Over-riding Constraints : Preference constraints can be used to order the parses produced by the parser. First those parses are seen which satisfy the preference constraint. However, if a parse does not satisfy the preference constraints, it is still produced, but only later.

Different strategies can be adopted to decide the preference constraints. One such strategy based on cost function decides on the basis of following :

1. Karta has the following preferences in descending order: human, non-human, inanimate. (animacy preference)
2. A source group is close to the demand group with which it has a relationship. (closeness preference)
3. Karta occurs before karma in a sentence. (leftness preference)

1.4 Our approach for karaka disambiguation:

For karaka disambiguation our approach uses the following facts taken from [5] :

Fact1: The sentences of Indian languages fulfill the akanksha-yogyata principle which states that each demand group imposes certain restrictions on the source word groups that appear as its karaka. Each source group should possess some yogyata in order to fulfill the demand of demand group. The most important demand groups are verbs with karaka demands, and the most important source word groups are the nominals.

Fact2: The karaka charts expressing restrictions as above are similar to sub-categorization. (If semantic types are also included then they include selectional restrictions.)

In our approach, we have made an attempt to provide information on the preferential constraints, used for karaka disambiguation. The ambiguous attachments can be resolved on the basis of the relative vibhakti distribution in the sentences of a large corpus. This approach describes a method for selecting suitable attachments using statistical data extracted independently from source language texts. The statistical data used here has been obtained by counting occurrences of vibhaktis of nouns in sentences with a given verb. The vibhakti-charts are constructed out of such statistical data.

1.5 Achievements and Contributions

sis, we have shown how corpus-based methods can be used in conjunction with ased methods. The following are the achievements:

1. We have used anusaaraka to perform morphological analysis of the given Kannada corpus, and to obtain vibhaktis of noun etc. In the process, we had to develop several filters to make corrections for participle, avyaya, kriya-mula, relational word, and samaasa etc. It should be mentioned that the use of anusaaraka shows that persons not knowing a language can still work on it. For example, we do not know Kannada and yet we were able to perform the analysis manually whenever required.

2. We have suggested two corpus-based methods to do statistical analysis and averaging of noun-vibhaktis for a verb in sentences in a corpus. The first method treats the occurrence of a given vibhakti (say 1) independent of others. The second method treats the vibhaktis as vector and uses the notion of vector distance. The analysis yields an average called vibhakti-chart which can be used in identifying attachments for complex sentences.

1.6 Organization of thesis

Outlined below are the chapter wise details of the various topics encountered while dealing with karaka disambiguation using vibhakti-chart approach.

Chapter 2 deals with the types of ambiguities. It discusses why ambiguities are particularly troublesome in NLP. It also discusses about the conventional approaches for its resolution.

Chapter 3 gives an overview of the corpus-based approach. It mentions about the usefulness of corpus-based approach in various sectors. Various corpus-based approaches for ambiguity resolution have been discussed. The chapter ends with a note on the disadvantages of corpus-based approach.

Chapter 4 deals with the system overview. It starts with the basic framework and explains the implementation details, and other related aspects.

Chapter 5 deals with the details of various filtering modules. It discusses general methodologies for filtering and explains each filtering case in detail, with examples.

Chapter 6 describes how the task of development of vibhakti-chart is accomplished. It also discusses the methodology for resolving ambiguities using the vibhakti-charts.

Chapter7 gives a summary of the work. It deals with the problem faced and suggests future work.

Chapter 2

NLP and Ambiguities

Absence of ambiguity is the most important property of any formal language. On the contrary, natural languages are social communication tools which constantly evolve and therefore are highly ambiguous and complex in nature.

Important properties of natural languages are :

- There is no limit to the number of different words.
- A word may have (and often has) more than one meaning.
- The number of phrases, sentences and other similar structures is potentially infinite, and exceptions abound.
- No sharp border exists between correct and incorrect constructions in actual use.
- Ambiguities are present at all levels of the language.
- It is hard if not impossible to give a complete syntactic description of a language. It is difficult to talk about absolute correctness of a sentence. A sentence can be distorted without hindering the understanding.

Further, in natural languages some words become obsolete and drop from use, whereas others are created or resurrected.

2.1 Ambiguities and NLP

Ambiguities are an integral part of NLP. The presence of ambiguities at all levels is the source of complexity of natural languages, and it constitutes a major problem for automatic language processing.

Ambiguities come in many varieties. Following are the major types :

1. **Lexical ambiguities :** It exists when a word has more than one meaning or belongs to more than one class. In such cases, the location of the word within a sentence (its syntactic use) will often permit the parser to discard most unwanted choices.

2. **Syntactic ambiguity :** A common syntactic ambiguity in Hindi is assignment of karta.

However, the parser tries its best to resolve the ambiguities. But no natural language processor exists that can perfectly interpret all possible statements. In fact, we can grade a natural language processing system by the degree to which it can successfully overcome the following sort of difficulties :

1. Multiple word senses
2. Modifier attachments
3. Noun-Noun modification
4. Pronouns
5. Ellipsis and Substitution
6. Anaphoric references
7. Ambiguous noun groups

2.2 Ambiguity Resolution :

We now outline different approaches that have been used for resolving the ambiguities as mentioned above.

2.2.1 Conventional Rule-Based or Grammar-Based Approach

In conventional methods, linguistic restrictions described in a dictionary and grammar are used to select a suitable translation. In general these restrictions are defined logically from the characteristics of another expression which modifies or is modified by the expression being processed. For example, to translate predicates (verbs and predicative adjectives), semantic restrictions are based on essential case arguments in the forms of semantic markers to indicate features of words or terms in the thesaurus to show a hierarchy composed of word concepts.

Though these conventional methods have been very useful in realization of natural language processing systems, they have many problems. Restrictions on all dependencies cannot be described in advance. The system suffers from an inability to select suitable translations if the input expression meets two or more restrictions, and have difficulty in processing any expression that violates the restrictions. Moreover, the descriptions of the restrictions is based on direct structural dependencies, therefore it is difficult to describe restrictions based on a sister dependency or between expressions belonging to different sentences or paragraphs.

2.2.2 Corpus Based Approach

Conventional methods for resolving ambiguities have shown no further improvements. However, other methodologies have been adopted in order to resolve more complex ambiguities and thus to improve the performance of the parser. In this direction *corpus based approaches* have become quite popular and are now in wide use. In the next chapter, we discuss it in detail.

Chapter 3

The Corpus-Based Approach

3.1 Introduction

Text corpus is a large body of collected text taken from sources of actual use which might be general or subject specific. Text corpora have proved to be important in numerous linguistic analyses over the last three decades. Research on the linguistic patterns of English by a number of American and European researchers has shown that linguistic analysis based on a collection of texts often do not conform to our prior intuitive expectations. The use of computer based text-corpora, together with computer programs to facilitate linguistic analysis, enables investigations of a scope not otherwise feasible. While corpus work has by no means been restricted to the English language, it is here that development has been most spectacular during the last fifteen years.

Corpus based studies constitute an important advance over previous research in that they are based on naturally occurring discourse, representing actual usage rather than linguists' intuitions. Computer corpora present the computational linguist with the diversity and complexity of the real language which is more challenging for testing language models than intuitively derived examples. Ultimately grammars must be judged by their ability to contend with the real facts of language and not just those discussed by grammarians.

Thus, researchers such as Francis and Kucera (1982); Johansson and Hofland (1989); Oakman (1975); and Ross(1973), etc., have combined computational analysis and computer-based text corpora to compare the linguistic characteristics of texts and text types.

With respect to lexicographic research, the result of this combination as applied in COBUILD project (Sinclair 1987), were so successful that now a number of publishers are pursuing dictionary projects derived from computer-based corpora using automated techniques.

Similarly, with respect to research on natural language understanding systems, researchers such as Garside, Leech and Simpson (1987) have combined corpus-based and grammar-based approaches to develop natural language understanding systems that are much more robust than previously achieved.

3.2 Corpus based approach for Ambiguity Resolution :

Recently many machine translation systems have been developed and put into practical use, but ambiguity resolution in translation is still one of the biggest problems in such systems. These systems have conventionally adopted a rule-based disambiguation method, using linguistic restrictions described logically in dictionary and grammar, but it is impossible to provide all the restrictions in advance. Furthermore, such systems have no reasonable means to select the most suitable translation if the input expression meets two or more restrictions, (that is, has two or more parses) or if the input expression meets no restrictions. Resolving syntactic ambiguity is still one of the biggest problem in machine translation systems.

In order to overcome these difficulties, the following corpus-based methods have been proposed:

Example Based Translation : This method is based on translation examples (pairs of source text and its translation) [17, 20]. This type of system called analogy-based or example-based machine translation, involves storing a large number of bilingual translation examples as a database, and translating input expressions by retrieving an example most similar to the input from the database. There is no failure of the output in this case, because it selects the most similar example not necessarily an identical one.

However example-based translation systems need a large database of translation examples. It is difficult to collect sufficiently large bilingual corpora into fragments and link them automatically. To overcome this problem a new mechanism based on sentential examples in a dictionary is used, which utilizes the merits of both the

translation by logical restrictions and the example-based methods, by selecting an equivalent translation which is most similar to the input expression. This mechanism can guarantee that it will always come up with some translations [11].

Structured Corpus Based Translation : This method uses structured bilingual corpora coupled together by cross coding translation units [16]. A method using syntactically and referentially structured bilingual corpora coupled together by cross coding translation units has also been proposed [16]. This bilingual corpora is called Bilingual Knowledge Bank (BKB) and is used as a knowledge base on various levels for machine translations and other applications. This kind of structured and cross-coded bilingual corpus is useful, but again it is difficult and expensive to collect, analyze and cross code.

Statistics Based Translation : This method uses statistical or probabilistic information extracted from large corpora [9, 18]. Statistical language learning is based upon two technologies . The first being the one that NLU community has developed such as programs for tasks like parsing sentences, assigning semantic relations to the parts of sentences, etc. Another being the probabilistic and information-theoretic foundation.

A statistical approach to learning a language would take a corpus, and learn the language by noting statistical regularities in that corpus [10]. Syntax has always been the item of major emphasis for analysis as most of the statistical work has concentrated on it.

Still, each has inherent problems and is insufficient for ambiguity resolution. For example, an example based translation system needs a large database of strict pairs of source text and its translation, and its difficult to collect sufficiently large bilingual corpora.

3.3 Disadvantages of Corpus-Based Processing

However there may also be potential hazards embedded in heavy dependence on corpus data alone.

- One danger is the convenient replacement of laborious hands-on analysis by rapid, automatic processing in many areas of linguistic study, which might miss out on important distinctions or worse do an incorrect analysis. Careful manual analysis can't be dispensed with.
- Another trap is the delusion that corpus size ('big is beautiful') is more important .
- A great risk is the distance that may arise between the end user of a standard corpus and the primary textual material. Usually the corpus is created by people other than the user, who may not be available for consultation. A corpus text may then be treated as a kind of canon.

Chapter 4

System Overview

In this chapter, overall scenario of the process is presented, and the implementation details have been discussed. It starts with a discussion on the basic framework.

4.1 Framework

As mentioned in section 1.4, our approach is based on vibhakti-chart. The overall process involves two major tasks, *viz.*

1. Development of vibhakti-chart, and
2. Use of vibhakti-chart for karaka-disambiguation.

Figure 4.1 shows an overview of the framework, used for this. The process consists of three phases:

Phase I: In the first phase the source text is analysed, using the output of anusaaraka (principally morphological analysis) and based on this analysis vibhaktis of nouns are extracted for each verb group in a sentence. A sentence is said to be a *simple sentence* if it has only one verb group, or no verb group (which occurs when the only verb is the 'be' verb and it is dropped); otherwise, it is treated as a complex sentence.

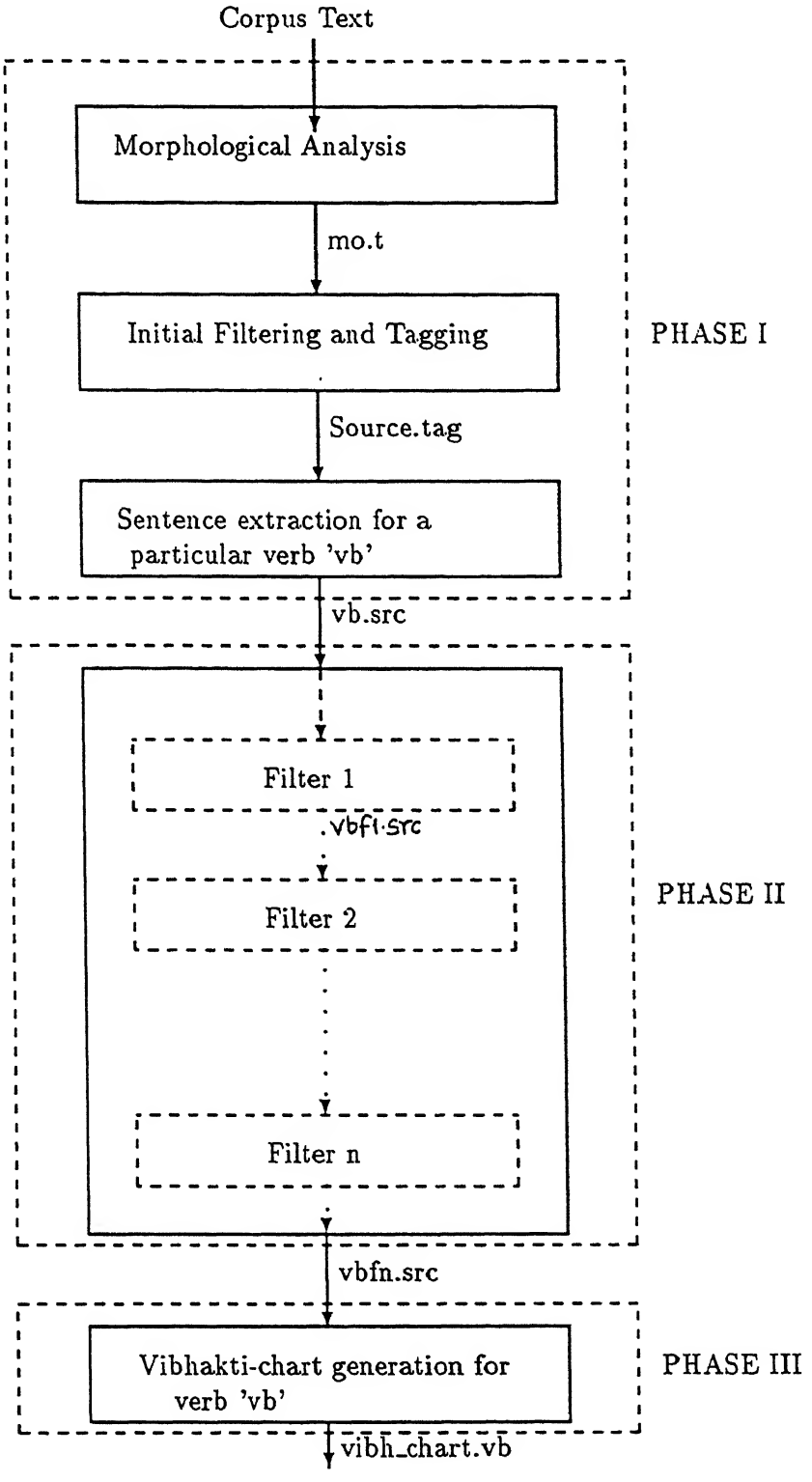


Figure 4.1: Overview of the framework

Phase II: In the second phase, the extracted text is filtered to remove or handle different problematic cases.

Phase III: In the third phase, vibhakti-chart is generated for each verb. It takes the output of PhaseII as its primary input.

Details of Phase II and Phase III have been discussed in chapter 5, and chapter 6 respectively.

During Phase I, the morph.'s output file *'mo.t'* is used for analysis and selection of simple sentences. Our module counts the verb occurrences in a sentence. If the verb count is either zero or one, then it is kept in the list of simple sentences for further use.

4.1.1 The corpus used:

Kannada story text corpus has been used for our work as it was readily available. It is a collection of more than 250 stories, and the total size is about 100000 words.

We used anusaaraka for understanding the Kannada sentence by obtaining its equivalent in anusaaraka Hindi. Since, anusaaraka follows the source text faithfully, it is ideally suited for such studies.

Chapter 5

Filtering

This chapter describes the details of filtering modules. Fig. 5.1 shows the steps involved in filtering and the corresponding inputs and outputs. Sections after 5.3 show different filters: what they try to achieve and how they achieve it. The implementation details are given in PERL language. (PERL uses regular expression notation standard in other utilities of UNIX. The notation is described in Appendix D.)

5.1 Introduction to Filtering

Due to language characteristics many a times analysis shown by morphological analyzer for various entities is not correct. Such situations warrant proper attention. Handling such situations can in general be termed as Filtering.
Filtering is comprised of the following two steps:

Step1: Determining whether a particular vibhakti of a noun is genuinely associated with the given verb or whether its apparent occurrence is due to some error. This is done primarily on the basis of language features and constraints.

Step2: Performing the corrective measures for vibhakti (as identified in Step1). A corrective measure can be of following types:

- Ignoring the sentence (having such occurrence cases) for the current demand group.
- Changing/modifying the occurrence for the correct vibhakti.

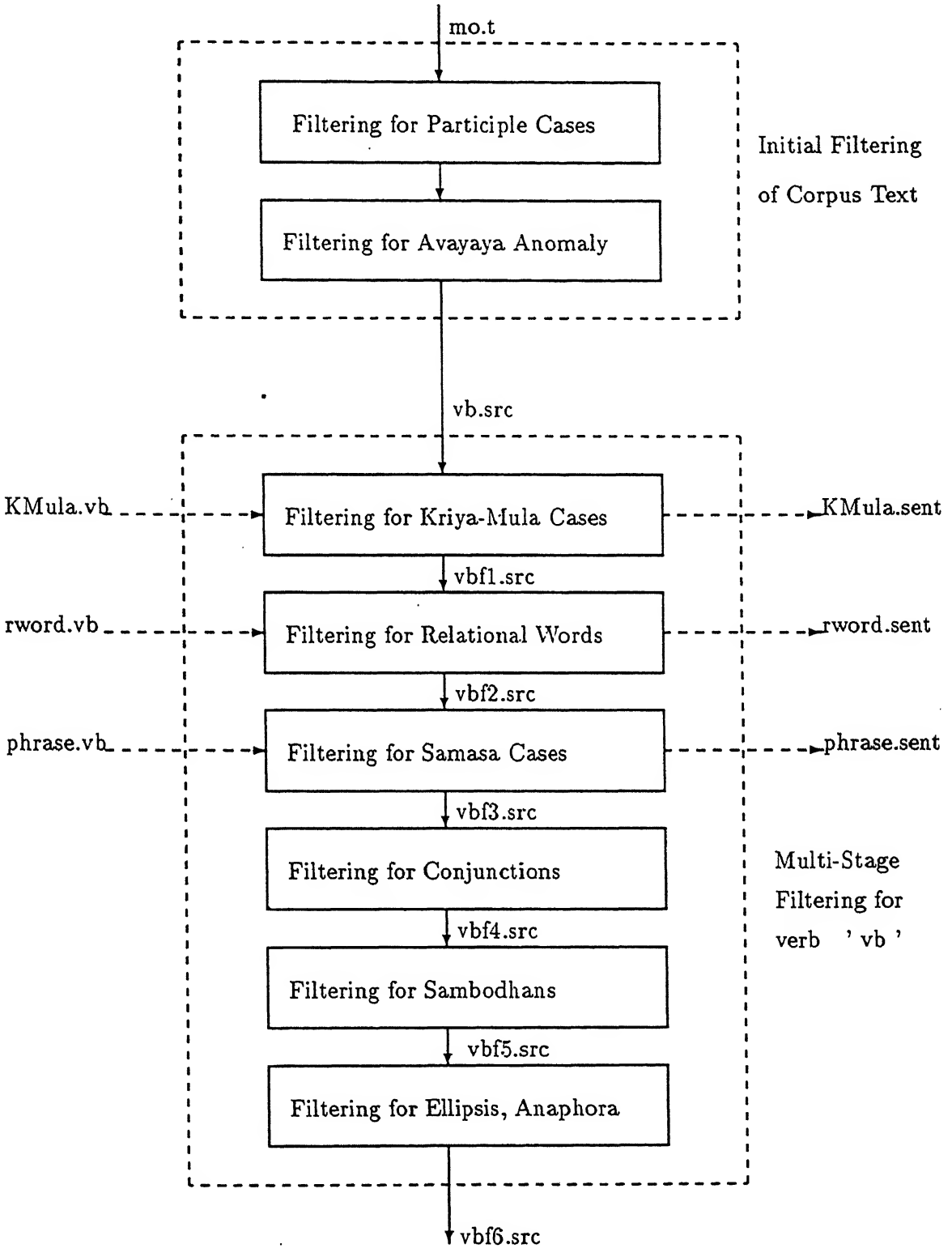


Figure 5.1: Filtering Stages.

- Removing/Deleting the vibhakti for such occurrence.

The Cases which demand filtering in particular are :

1. Kriya-mula cases
2. Relational word cases
3. Conjunction cases
4. Samaasa
5. Participle cases
6. and others as discussed.

All such cases have been explained in detail in the upcoming sections.

5.2 Methodologies of Filtering

Filtering process involves the following :

- Identifying the filtering case
- Implementation of corrective measure
- Production of the corrected output

Based upon these, filtering can be of three types for a particular case *viz.*:

1. Fully automatic
2. Semi automatic
3. Fully manual

Fully Automatic Filtering : In fully automatic filtering, the computer routine identifies the case and based upon the case, performs the desired corrective measure and produces the output. Cases of Kriya-mula have been handled with this filtering approach.

Semi Automatic Filtering : In Semi automatic filtering, for a given case, the computer routine helps in identifying the sentences where this case applies and needs the corrective measures. The information files produced by the routine contains the information

of sentences in which the particular case exists , and thus requires filtering. The user then manually corrects the vibhakti requirements/features for that sentence. As the manual intervention is needed, it is called semi automatic filtering. Cases of Relational Word are handled with this filtering approach.

Fully Manual Filtering : In fully manual filtering the identification as well as correction etc. are done manually. It is particularly useful for the cases of Metaphor and Anaphoras.

5.3 Initial Filtering

Initial filtering is done on the output file produced by the morph. It is done at this stage because such filtering is a general requirement, and is not particular to a given verb.

Following cases are taken care of during Initial Filtering :

- Participle cases
- Avyaya anomaly cases

5.4 Participles and the Absolutive

Participles are verbal-adjectives that modify a noun (or pronoun) but retain some properties of verbs. Therefore, these can be treated as modifiers formed from verbs.

calatA (walk), KAtA (eat), paDatA (read), AtA (come), KAyA (ate), paDA (read), etc. are some examples of participles.

The absolutive is formed by combining the verb *'kar'* with the root form of the main verb. *jA_kar (having gone), KA_kar (having eaten), so_kar (having slept), etc.*, are examples of the absolutive.

5.4.1 Handling Participles and Absolutive

The present participle is an adjective derived from a verb by appending TAM label *'wA'*, such as *calawA, AwA etc.* Past participle is formed by appending TAM label *'A' or 'yA'*.

Therefore, it is the TAM label that tells about the occurrence of participles and the absolutive.

A part of the file used by morph for TAM analysis has been illustrated here for observation.

"nf1","0_kara",

"nf4","0_kara BI",

"nf5","0_kara hI",

"nf6","0 rahA",

"nf7","0 wA_WA",

"nf8","wA_huA",

"nf9","wA_huA/wA",

"nf10","wA_huA samaya",

"nf11","wA_rahA",

"nf12","wA hI",

"nf15","wA_WA vEse/wA_WA Ese/wA hI",

"nf19","yA_huA",

"nf26","yA_huA/0_liyA huA",

"nf37","nA",

"nf38","yA_hE",

"nf69","nA se/0_kara",

From this, it is evident that TAM labels for participle and absolutive cases have been coded as "nf[0-9]+". We have used this feature in our module for identifying the sentences having participle cases.

5.4.2 Implementation Details

While processing the input file, the TAM label for each verb is checked for the pattern string nf[0-9]+. If the pattern matches, it means that verb has been used in the participle form in the sentence, and the sentence must be treated as a complex sentence. It must not be considered as an example of a simple sentence.

Implementation

Input : Source file **mo.t** from morph., of anusaaraka

Output: Finally the file **baru.src**

Filtering type: **Fully automatic**

Algorithm :

```
Repeat foreach sentence "s" in the standard input stream
  Foreach verb "vb"
   if TAM lebel of "vb" matches " nf[0-9]+" then
   don't consider the sentence "s" to be a simple sentence.
  endfor
endrepeat
```

5.4.3 Further Cases

1. 'wA' forms of verb usually need very careful analysis.

   rama phal KAtA hE.
   phal KAtA huA ladakA rone lagA.

   In the above sentences word *KAtA* has been used as a verb. But in the following sentence the word *KAtA* refers to a noun *(ledger)*

   jab se KAtA huA hE nIMda nahIM AtI.

2. The present participle can be used as an ordinary Adjective :

   bahtA pAnI sApha hotA hE.
   Kilte PUloM ko mata todo. Occasionally, however a *huA, huI or hue* is placed after the present participle.

   bahwA huA pAnI sAPa hotA hE.
   Kilte hue PUloM ko mata wodo.

3. A present/past participle, can be used as a noun, also.

   dUbate ko bacAo.
   vaha rotoM ko haMsAtA hE.

maroM ko mata mAro.

paDe-liKe ko kyA samaJAyA jAye.

4. Pairs of allied verbs can form a 'compound absolutive'.

   KA-pI kar , paDa-liKa kar etc.

5.5 Avyaya anomalies

Many words like *vahAz_se , yahAz_se etc.*, do involve vibhakti "5", but are being shown by morph., as avyaya. Morph. considers a word to be an avyaya, if it does not change its form. Since, morphologically *vahAz_se , yahAz_se* are also invariant so these are being treated as avyayas. So there is a need for correcting the cases of such occurrences. Following example illustrates a case of such occurrence.

```
baru; 5,1:   !![story108,35]
!!alliMxa{5*} puswaka{1} baru{v}
 alliMxa 200 puswakagalVu  baMxavu.
 <31> vahAz_se 200 puswakeM AyA.
```

The '*' mark in the tag of *alliMxa{5*}* indicates that our module has corrected it for the vibhakti 5.

5.5.1 Handling Avyaya Anomalies :

It is to be noted that different avyayas might be involving different vibhaktis. Following vibhaktis are reflected by the avyayas as indicated

1. *ko - kahAz_ko,yahAz_ko,Age_ko*
2. *se - kahAz_se,vahAz_se,Age_se*
3. *kA - kahAz_kA,jahAz_kA,jab_kA*
4. *par - yahAz_par,jahAz_par*

5.5.2 Implementation Details

A file Cross.avy contains all such words which in fact should have been considered as noun and not as an avyaya. An avyaya in consideration is checked for such avyaya cases and if it matches then the corresponding vibhakti is recorded instead of considering it to be an avyaya.

Implementation

Input : Source file **mo.t**

Output: Finally the file **baru.src**

Filtering type: **Fully automatic**

Algorithm/code :

```
# (Note that all code is in the popular PERL language.)
$key = "\"".$wd."\"";
open(FL,"../Lib/Cross.avy") ;
while(<FL>) {
      if ( /^$key/) { if( $_ =~ /#/)
      {
           $msg_A=substr($',0,1); #
           $t_msg_A = $msg_A."*";}}}
```

5.6 Kriya-Mula :

Kriya-mula constructions are sequences of nouns or adjectives plus a (light) verb which are viewed semantically as units functioning as a verb. By functioning as a unit we mean that there is a single demand-chart (karaka-chart) for the entire unit. There is no demand-chart for the light verb alone.

Some examples of kriya-mulas are :

kshyamA karnA (to forgive), yAda AnA (to remember), diKAI denA (to come to view), burA laganA (to dislike), acCA laganA (to like), etc.

In order to know the characteristics for identification of kriyA-mulas in the sentences, let us take the following example sentence

isase akbara_ko' bahuta kopa AyA. (By this Akbar got very angry.)

In this, there is a combination of noun *kopa*(anger) and verb *AyA* (come+past). Here *AnA*(come) is not the main verb but *kopa-anA* (to get angry) itself is the main verb phrase that denotes the single verbal idea of *getting angry.* Therefore, there is an occurrence of a kriya-mula.

Since kriya-mulas behave semantically as verbs, their karaka requirements need to be satisfied. But since the karaka requirements of kriya-mula are not the same as that of verb involved in it, the karaka chart of the verb involved is transformed appropriately depending on the noun or adjective present to obtain the karaka chart of the kriya-mula.

5.6.1 Handling Kriya-mulas :

Kriya-mulas may be the major source of wrong information. This has been illustrated through the following example.

```
baru; ,4,1_6:   !![story189,9]
!!SrImaMwa{4} kopa{1_6} baru{v}
  SrImaMwanigeV kopa baMxiwu .
<20> XanI_ko' gussA AyA.
```

Here, any morphological analyzer would suggest the main verb to be *baru*, which in fact it is not. Therefore this sentence should not be considered as an example of the verb *baru*, and must be deleted from the corpus of this verb.

5.6.2 Implementation Details :

For a given verb *'vb'* a file *KMula*.vb, contains the list of noun and adjectives whose combination leads the verb *'vb'* in formation of a kriya-mula. The filter for kriya-mula checks the corpus for those sentences which involve kriya-mulas; identifies and deletes these from the output.

Implementation

Input : Source file **baru.src**

Output: Files **baruf1.src** and **KMula.sent.**

Uses : Kmula.baru

Filtering type: **Fully automatic**

Algorithm :

```
#(The first module) :
cp baru.src tmp1
for i in 'cat KMula.baru'
do
 echo $i
 cat tmp1 | grep -v $i > tmp2
 mv tmp2 tmp1
done
 mv tmp1 baruf1.src

#(The next module) :
cp baru.src tmp1
rm KMula.sent
for i in 'cat KMula.baru'
do
 echo "--------------" >> KMula.sent
 echo $i >> KMula.sent
 echo "               " >> KMula.sent
 cat tmp1 | grep  $i >> KMula.sent
done
```

5.6.3 Further Cases

1. The verbs most frequently combining with a noun or an adjective and thus forming kriya-mulas are *karanA, honA, AnA, paDanA and laganA.* Some examples :

mEMne cora ko kshamA kiyA. (I forgave the thief.)

pAtha AraMBa karo. (Begin the lesson.)

ladakI ko lajjA AI. (The girl felt ashamed.)

use roja buKAra AwA hE. (He gets fever everyday.)

bacce ko Buka lagi hE. (The child is hungry.)

muJe Sora burA lagawA hE. (I dislike noise.)

2. *honA, AnA, laganA and rahanA* form Intransitive compounds usually by combining with nouns. Most of these compounds are of a passive nature although of active formation.

3. A list of detailed examples of Kriya-mula for *karu* has been illustrated here -

   sanxeh AnA (to doubt); gussA AnA (to get angry); vicara AnA (to get an idea); haMsI AnA (to feel laugh); najar AnA (to come in sight); uwsAha AnA (to feel encouraged); uwwar AnA (to have a reply); nIndrA AnA (to feel to go to bed); sankat AnA (to face trouble); vixyA AnA (to get knowledge); kopa AnA (to get angry); upyoga meM AnA (to come in use) ; buKAra AnA (to get fever); varRA AnA (to have rain); hoSa AnA (to get senses); XayA AnA (to feel pity); lajjA AnA (to(begin to)feel ashamed); yAxa AnA (to recall); etc.

5.7 Relational Words

A word that appears in the sentence after a noun and depicts a relation of the noun with other words of the sentence, is called a relational word. Let's consider the following sentences :

nOkara gAvoM taka gayA. (Servant went upto the village.)

rAta Bara jAganA acCA nahIM. (Waking up whole night is not good.)

Here *taka, Bara* are relational words as they are defining relation of *gAvOM* and *rAta* to the other words, respectively.

Some other different examples of relational words are

sabase *pahale* (before anybody else), Sahar se *dUra* (far of town), samUdra *pAra* (across the sea) etc.

```
open( OUT1, ">rword.sent");
@REL = <RW>;
@SOURCE = <SRC> ;
foreach $c(@REL){
        chop($c);   print $c ;
        @foo = grep(/$c/,@SOURCE);
        foreach $elm(@foo) { print OUT1 $elm ;}}
```

5.7.3 Further Cases

1. Relational words usually follow noun, but *mAre, binA, and sivA* are used sometimes for quantification and negation, before the noun which they govern.*e.g.*,

   binA uske (without him);

   mAre Buka ke (on account of hunger);

   sivA mere (except me)etc.

2. Relational words such as *Age, pICe, wale, binA etc.*, sometimes appear with omissions of case signs.

   naxI pAr *naxi(ke) pAr* (across the river)

   pITa pICe *pITa(ke) pICE.* (behind the back)

3. Relational words *pare, rahiwa* always come with case sign *"se"*, while words like *pahle, pICe, age, bAhara etc.* come alternatively either with *"se"* or *"ke"*.

   samaya ke pahle *samaya se pahle.* (before time)

5.8 Samaasa

Samaasa is a word formation in which more than one nouns are combined together. Usually a '-' is used between the words. *e.g.*,

putra-ratna, dahI-badA, vana-mAnuSa etc.

- Words formed by the repetitions of a particular word have also been considered in the category of samaasa.

1. repetition of noun : *pAnI-pAnI; galI-galI; bUMda-bUMda*
2. repetition of pronoun : *apanA-apanA; kisa-kisa; kuCa-kuCa*
3. repetition of words with post-positions : *gAvoM_ka_gAvoM*

- Words formed by combination of two phrases have also been considered in this category.

  swarga-prApta; AcAra-kuSala; akAla-pIdita; rasoi-ghar; mAla-godAma; kAma-cOra; ghodA-gADI; rAma-kahAnI; gaMgA-jala etc.

5.8.1 Handling Samaasa :

If the samaasa appears in a sentence without the hyphen mark (-) then the constituent parts of the samaasa, are considered as separate words by the morph. Due to this the vibhakti requirements exhibited by it can not be considered as such. The following example illustrates the above mentioned problem.

```
baru; A,1,1,1:   !![story34,20]
!!bahalVa{A} kAla{6} naMwara{A} aWarva{1} vexa{1} baru{v} eVMba{A}
  prawIwi{1} ixeV{A}
  bahalVa kAlaxa naMwara aWarva vexa  baMxiwu eVMba prawIwi ixeV .
<92> bahuwa samaya_kA bAxa_meM aWarva vexa AyA EsA prawIwi hE.
```

Here "aWarva vexa" is supposed to be considered as one phrase, instead of two nouns. because this has happened due to the separate consideration of the constituent parts of the samaasa. The filter makes the appropriate changes to it. Accordingly the vibhakti string 'A,1,1,1' is modified to 'A,1,1'.

Further it is also possible that the constituent parts may carry entirely different meaning for the sentence when they are considered as entities in isolation. In such cases, they are supposed to be considered as separate entities and their individual vibhakti is required to be recorded.

Some examples of such cases are illustrated :

```
ghodA-gADI sundar hE. (Horse-cart is beautiful.)
ghodA gADi KIncatA hE. (Horse pulls the cart.)
```

```
AkASa-sancAr bahuta unnatI kar gayA hE.
(Sky-communication is in high progress.)
AkASa sancAr ke liyE upayoga hotA hE.
 (Sky is used for communication.)
```

5.8.2 Implementation details

A File consisting of all the sentences that involve samaasa is generated. The user is expected to look at these sentences and classify correctly.

Implementation

Input : Source file **baruf2.src**

Output: Files **baruf3.src** and **phrase.sent**

Uses : File **phrases.baru**

Filtering type: **Semi automatic**

```
open( SRC,"baru.src");
open(RW,"phrases.baru");
open( OUT,">baruf3.src");
open( OUT1, ">phrase.sent");
@PHRASE = <RW>;
@SOURCE = <SRC> ;
foreach $c(@PHRASE){
        chop($c); print $c ;
        @word = split(/-/,$c) ; # breaks the phrase in to constit. words
                                # key is "-"
        print $word[0],$word[1];
  foreach $x(@SOURCE) {
          if (($x =~ /$word[0]/) && ($x =~ /$word[1]/)) {print OUT1 $x;}}}
```

5.9 Ellipsis and Anaphora Cases

Ellipsis means omitting the constituent from a sentence. This is usually done when the constituent has occurred in an earlier or the preceding sentence. Thus incomplete sentences

or drop out cases are called cases of ellipsis.

In the following example there is ellipsis in the underlined portion.

```
baru; A,1,1_6,1:    !![story31,4]
!!SrAvaNa{1} mAsa{1_6_} hAgU{A} paMcami{1} hawwira{A} baru{v}
  SrAvaNa mAsa  nAgacawurWi hAgU paMcami hawwira baMxiwwu .
<53> SrAvaNa mAsa[gaMxagI_kA] %sarpa_cawurWI evaM paMcamI pAsa AyA_huA_WA.
     SrAvana mAsa sarpa_cawurWI evaM paMcamI pAsa AyA_huA_WA.
```

Anaphoric reference exists where a sentence or word makes a reference that can be interpreted only in terms of earlier sentences, phrases or words. For example, in the following ten given sentences, anaphoric reference does exist. It is evident from the fact that *usako'* in sentence 2, refers to *siyAra*, which is evident only from sentence 1.

```
1.eka jaMgala_meM eka siyAra WA.
2.usako' eka xina bahuwa pyAsa huA.
3.isaliye vaha pAnI_ko Koja_kara nikalA.
4.KojawA_huA KojawA_huA vaha eka bAga_ko' AyA.
5.bAga_mEM eka kuAz WA.
6.siyAra usakA aMxara kUxA Ora pAnI_ko pi_kara
  pyAsa_ko nivAraNa_kara_liyA.
7.lekin usako' Upara caDanA_* sAXya nahIM_huA.
8.WodA samaya _huA Uprara eka bakari vahAz AyA.
9.usako' Bi bahuwa pyAsa huA.
10.vaha aMxara Juka_kara xeKA.
```

Anaphora and pronouns require antecedents, that must be bound for the correct translation. These find their antecedents in the karta of the sentence, unless anomaly is there. In our approach, anaphora and ellipsis are being handled purely on the manual basis.

5.10 Other Filtering Cases

Followings are some of the important cases, that in general demand modifications as mentioned :

- Occurrences of multiple nouns combined together with the conjunction *Ora* (and) should be treated as single occurrence.

  rAma,mohana Ora harI ghara aye. (Ram, Mohan and Hari came to the home.)

  In this sentence morph., will provide vibhakti '1' for each noun *viz.* rAma, mohana and harI. However, the correct vibhakti pattern should contain only '1'. In our approach, we are achieving it on manual basis.

- "Sambhodhan" words should not be considered as part of the input sentence.

```
baru; A,1_6,1:    !![story117,9]
!!amma{1_6_} illigeV{A} oVbba{A} manuRya{1} baru{v}
   amma , illigeV oVbba manuRya  baMxixxanu .
<70>  mAz , yahAz eka manuRya AyA_huA_WA.
```

  Here the sentence *'mAz , yahAz eka manuRya AyA_huA_WA. '*, should be modified as *'yahAz eka manuRya AyA_huA_WA.'* , and accordingly the vibhaki string *'A,1'* should be provided, instead of *'A,1_6,1'*.

  But in the following sentence there should not be any modification as, here "amma" has not been used as sambodhan.

```
baru; A,1_6,1_v_v_1,4:    !![story191,5]
!!amma{1_6_} bahuSaH{A} Aru{1_1_} gaMteV{4} baru{v_v}
  amma : bahuSaH Aru gaMteVgeV baMxenu .
<72> mAz : SAyaxa Ce[*TaMdA_ho*] GaMte_ko' SAyaxa_AegA.
```

  We are handling the sambhodhan cases on manual basis.

- Most adjectives are shown as avyayas by the morphological analyzer. But adjectives such as *mithA, mithAsa, lambA, CotA, UncA, nIMcA etc.*, are shown as noun instead of adjective. It is because these have variant forms. One such example is :

  bahuta mithAsa ho_kar[ho] bhi hE. (also with a great sweetness)

In our approach sentences are being searched manually for such cases and the modifications are made accordingly.

Chapter 6

Vibhakti Chart

This chapter describes how the task of development of vibhakti-chart is accomplished. The chapter also discusses how vibhakti-chart can be used for resolving attachment ambiguities.

6.1 Vibhakti-Chart

A vibhakti-chart for a given verb gives the probability of occurrence of each of the vibhaktis with the given verb in a sentence.

6.2 Format of a Vibhakti chart

The general format for a vibhakti-chart is shown in figure. 6.1.

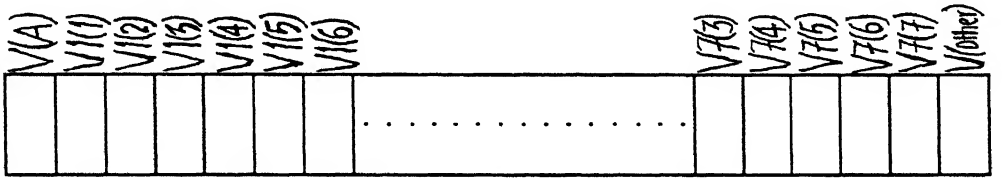


Figure 6.1: Format of a Vibhakti-chart

In the figure 6.1 various elements are as follows:

V(A) : represents for the occurrence of avyaya.

V1(1) : represents the probability of occurrence of vibhakti 1.

V1(2) : represents the probability of occurrence of vibhakti 1,1.

V1(3) : represents the probability of occurrence of vibhakti 1,1,1.

.

.

V7(6) : represents the probability of occurrence of vibhakti 7,7,7,7,7,7

V7(7) : represents the probability of occurrence of vibhakti 7,7,7,7,7,7,7.

V(other) : represents the probability of occurrence of vibhakti other than 1-7.

6.3 Steps for generation of a vibhakti-chart :

For a particular verb, generation of vibhakti-chart involves the following steps using a corpus:

Step1: Extraction of the vibhakti string for all sentences of the verb, obtained from the filtering phase. (Extraction of vibhakti string)

Step2: Generation of vibhakti-vectors for all vibhakti strings. (Generation of vibhakti-vector)

Step3: Developing a single vibhakti-chart out of all vibhakti vectors that were obtained in Step2. (Development of vibhakti-chart)

Each of the above step can be explained in the following way -

6.4 Extraction of vibhakti string (Step1)

Vibhakti strings of all the filtered sentences of the verb in consideration are extracted and kept in a separate file. For a verb *'vb'* this file is named as *vibh_string.vb*. Let a vibhakti string for a sentence S1, look like *A_1,1_v,1_4_3*. This vibhakti string conveys the following information :

- Three nominals appear in the sentence S1. (because there are 3 units separated by commas.)
- First nominal (corresponding to A_1), is either an avyaya or has vibhakti 1. (This is due to ambiguity in lexical category of the word.)
- Second constituent (corresponding to 1_v), is either a nominal with a vibhakti 1 or is a verb.
- Third nominal (corresponding to 1_4_3), has either vibhakti 1, 4 or vibhakti 3.

6.5 Generation of vibhakti-vector (Step2)

For each vibhakti string, mentioned in previous step, a vibhakti-vector is generated. The vibhakti-vector has exactly the same format as shown in figure 6.1. The vibhakti-chart in fact is nothing but the representative vector for all the vibhakti-vectors.

The module corresponding to generation of vibhakti-vector takes the vibhakti string (*e.g.*, A_1,1_v,1_4_3) as the only input. It models the string appropriately using the statistical methods, performs the computation as required and generates the vibhakti-vector.

As an example, for the generation of vibhakti-vector, let us take a case of simple vibhakti string A_1,4_1,1. The possible combinations for this are:

A,4,1

1,4,1

A,1,1

1,1,1

Here out of four possible combinations, 'A' is appearing in two combinations, so its probability of occurrence is 0.5 . Similarly, because Vibhakti '1' is appearing in all the combinations so its probability of occurrence is 1.0 . Vibhakti '1,1' is appearing in 3

combinations so its probability of occurrence is 0.75. Vibhakti '1,1,1' is appearing in only 1 combination so its probability of occurrence is 0.25, and Vibhakti '4' is appearing in 2 combinations so its probability of occurrence is 0.5.

Accordingly the vibhakti-vector is assigned the following value:

V(A)=0.5; V1(1)=1.0; V1(2)=0.75; V1(3)=0.25; V4(1)=0.5;

Other components of the vector have zero value.

6.6 Development of vibhakti-chart (Step3)

A vibhakti-chart can be developed in two ways. One uses the average of vibhaktis independently, while the other is based on distance function. The sections that follow deal with these approaches.

6.6.1 Vibhakti-chart based on average of vibhaktis treated independently

In this approach each vibhakti is considered independently. For a particular vibhakti, average is calculated for all of its values, shown in different vibhakti-vectors. The vibhakti-chart can thus be achieved by considering the average values of all the vibhaktis individually. (This may be referred as average vibhakti-vector also.)

6.6.2 Vibhakti-chart based on vector average

For generation of vibhakti-chart, using the concept of vector average following algorithm is used.

1. Find vibhakti-vector for all vibhakti strings.
2. Assume an initial vibhakti-string as the vibhakti-chart to be known as $\vec{V_{chart}}$. (A prime numbered vibhakti-vector or the average vibhakti-vector may be a good choice.)
3. Notion of distance between two vibhakti-vectors can be given as follows :

   (a) Scalar distance (D_{V_1,V_2}) between two vibhakti-vectors V_1 and V_2 is defined as :

   $$D_{V_1,V_2} = w_A * |\Delta V(A)| + w_1 * |\Delta V1(1)| + w_2 * |\Delta V1(2)| + ... + w_{other} * |\Delta V(other)|$$

   i.e.,

   $$D_{V_1,V_2} = \sum_i w_i * |\Delta V_i|$$

where *'i'* is the vibhakti and w_i corresponds to weight for vibhakti *'i'*.

(b) Vector distance ($\vec{D_{V_1,V_2}}$) between two vibhakti-vectors V_1 and V_2 is defined as :

$$\vec{D_{V_1,V_2}} = w_A * \Delta V(A) + w_1 * \Delta V1(1) + w_2 * \Delta V1(2) + ... + w_{other} * \Delta V(other)$$

i.e.,

$$\vec{D_{V_1,V_2}} = \sum_i w_i * \Delta V_i$$

Based on the definition of distance above, calculate the average distance *(avg_dist_chart)* of N vibhakti-vectors from $\vec{V_{chart}}$.

4. Correct the $\vec{V_{chart}}$ in following way :

   - Assume $\vec{V_{acc}} = 0$.
   - **Do** for each of N vibhakti-vectors $(V_1, V_2, ..., V_n)$

     If $D_{V_i,V_{chart}} > avg\_dist\_chart$ then

     $\vec{V_{acc}} = \vec{D_{V_i,V_{chart}}} + \vec{V_{acc}}$

     enddo

     $\vec{V_{chart}} = \vec{V_{chart}} + \vec{V_{acc}}$

6.7 Resolving attachment ambiguities using vibhakti-chart

The following algorithm can be used to resolve the ambiguities related to the noun and verb group attachments in a machine translation system.

1. Make a list of potential candidates for the ambiguity under consideration.
2. Check whether all these candidates can be accommodated in the target language sentences by appropriate choice of words. If not, divide the candidates in groups among which disambiguation is necessary.
3. Apply plausibility test to each group and award ranks depending on the vibhakti-chart information (probability of occurrence) for a member of the group.

The algorithm can further be augmented with information-theoretic concept of mutual information.

As explained in the previous sections, each element of vibhakti-chart depicts the probability of occurrence for that vibhakti. These probability values can be used to find out the mutual information for the members.

According to transmission of information in a channel, when two words x and y have the probabilities *P(x)* and *P(y)*, the mutual information $I_g(x,y)$ is defined to be :

$$I_g(x,y) = \frac{P_g(x,y)}{P(x) * P(y)} \tag{6.1}$$

The subscript g indicates the grammatical function of the word x to the word y. If there is a genuine association between x and y, then $P_g(x,y)$ will be much larger than *P(x)P(y)*. By definition $I_g(x,y)$ will be much larger than zero in this case. Similarly, if there is no interesting relationship between x and y, then the joint probability $P_g(x,y)$ will be almost equal to *P(x)P(y)* and thus $I_g(x,y)$ will be almost equal to one. (If x and y seldom occur together then $P_g(x,y)$ is almost zero and so is $I_g(x,y)$.) When the attachments of two words x and y is possible, mutual information for the word pair shows how strongly they are connected with each other. By definition, bigger $I_g(x,y)$ represents the stronger semantic relationship between x and y. Therefore when $I_g(x,z) > I_g(y,z)$, the semantic relationship of x and z is stronger than that of y and z.

Suppose, it is ambiguous whether phrase z is attached to phrase x or y, and $I_g(x,z) > I_g(y,z)$. It is clear that the attachment of z to x has a greater probability than the attachment of z to y. Therefore, it is reasonable to attach z to x in this case. However, the concept of mutual information is true, only when distinct vibhaktis are taken into consideration.

6.7.1 Illustration of Ambiguity Resolution

As an illustration of resolving attachment ambiguities with our approach we take two cases, one with no lexical ambiguity and another with lexical ambiguity as well.

In the first case, let a sentence contain two verbs v_a and v_b, and its nominals are appearing with vibhaktis 1,1,3 (*i.e.,* vibhakti string is 1,1,3). Now, it is required to convey information about the correct noun-verb attachments.

This can be achieved by following the undermentioned steps:

1. The assignments of 3 vibhaktis 1,1,3 and two verbs v_a and v_b can have the following distributions:

	v_a	v_b
(a)	1,1,3	-
(b)	1,3	1
(c)	3	1,1
(d)	1,1	3
(e)	1	1,3
(f)	-	1,1,3

2. Now it is required to calculate the probability of occurrence for each distribution. For example, the probability of occurrence for distribution (c) would be
$(1 - V_a1(1)).(1 - V_a1(2)).(V_a3(1)).(1 - V_b1(1)).(V_b1(2)).(1 - V_b3(1))$,
while for (f) it would be
$(1 - V_a1(1)).(1 - V_a1(2)).(1 - V_a3(1)).(V_b1(1)).(V_b1(2)).(V_b3(1))$,

3. The distribution for which the calculated probability is the highest, is to be treated as the correct karaka assignment.

As a second case, lets consider that the vibhakti string is appearing with lexical ambiguity as well. If the problem sentence has the vibhakti string'1_4,3,5', then we would have the following assignment distributions:

	v_a	v_b
(a)	1,3,5	-
(b)	3,5	1
(c)	1,5	3
(d)	1,3	5
(e)	1	3,5
(f)	3	1,5
(g)	5	1,3
(h)	-	1,3,5
(i)	4,3,5	-
(j)	3,5	4
(k)	4,3	5

(l)	4,5	3
(m)	3	4,5
(n)	4	3,5
(o)	5	3,4
(p)	-	4,3,5

In the same fashion as described in the first case, the probability of occurrence for the distributions (a)-(p) are calculated. The distribution with maximum probability is selected, which not only resolves karaka disambiguation, but also the lexical disambiguity.

For resolving ambiguities using the vibhakti-chart developed on the basis of vector average, the scalar distance for the given sentence is calculated with respect to vibhakti-charts of the verbs v_a and v_b, and then correct assignment is decided.

6.8 Discussion:

Since, the vibhakti-chart contains statistical information for vibhaktis, obtained from the actual text corpus, it can be used for many other applications, where vibhaktis play a crucial role. Some of the identified applications where vibhakti-charts can be used are :

- to find agreement rules for verb and noun-groups with proper vibhaktis.
- to find, how many noun-groups have ambiguous grouping with or without vibhakti-charts.
- to know whether verbs obtained can be grouped into classes.

Chapter 7

Conclusions

In this thesis we have shown how filters can be used to pass the corpus through a morphological analyzer, and to clean the data by passing through suitable filters to suit the purposes for which data is being gathered. Next, we have used this 'cleaned' data to obtain vibhakti-charts which are a subset of karaka-charts. These in turn are important in karaka disambiguation.

It is hoped that vibhakti-charts can be used effectively for resolving noun-verb attachment problems in Indian languages. Further the vibhakti-chart can be used for finding agreement rules for verb and noun-groups with proper vibhaktis; for finding how many noun-groups have ambiguous grouping with or without vibhakti-charts, and also to know whether verbs obtained be grouped into classes.

One may face the following problems, while adopting corpus-based approach for generation of vibhakti-charts:

1. A large amount of text in a specified domain is required which is not always available.

2. Text to be fed to an extraction program often needs to be tagged or annotated. This tagging is a time and resource consuming task.

3. A problem related to filtering is that, the total number of modules required can't be predicted in advance, because it is a continuous development process. Our approach favours a system architecture in which filters can be modularly added or removed.

7.0.1 Future Work

As a further extension to the work, following is suggested:

- Strategies should be explored to achieve fully automatic or semi automatic filtering for the cases that at present are being handled manually.
- Rigorous mathematical foundations can be developed for the disambiguation method based on vector average.
- Larger Kannada corpus of 3 million words completed recently by CIIL, Mysore and Department of Eletronics, Govt. of India can be used.
- Large scale testing of the efficacy of vibhakti-charts needs to be carried out. This can be done with a large corpus (of say a few million words).

Appendix A

A.1 Vibhakti strings for the verb *baru (come)*

,1

,1,1,1

,1,1_4

,1,4

,1,1

,1,1_6

,1_v,1,1

,3,1

,4

,4,4

,10,1

,4,1

,4,1

,4,1

,1

,7,1,3,1,11

,7,1_6

,1

5,1

5,A,A,4,1

A,

A,1

A,7,1,1

A,1,1

A,12

A,1_6,1_v_1,4

A,42

A,A,1

A,A,1,1_6,1

,A,1,1,3

A,A,1,4,4

A,A,1_4,1

A,A,1_4,1

A,A,1

A,A,1_6,1,3

A,A,4,1

A,A,4,1

A,A,7,1

A,A,7,1_6

A,A,5,1,1,8

A,A,A,1,1

A,A,A,1

A,A,A_6,1,3

A,A,A,1,7

A,A,A,A,A_v,1

A,A,A,A,v,1,1

A,A,v_v,1

A,A_1_v,1,4,1

A,A_v,1,1,2

A_1,1

A_1,A,4

A_1,A,7,4_4,3,1_v

A_1_v,1,1_6

A_1_v,A,4,1

A_1_v,A_4,A,A,3,1,1

Appendix B

B.1 Vibhakti-vectors for the verb *baru (come)*

-,1,0,0,0

A,0,0,0

A,1,0.6,0.1,0,0,0

Appendix C

C.1 Vibhakti-chart for the verb *baru (come)*

A,.94,.85,.73,.51,.28,0.0,.29,.24,0.0,0,0,0,0,0,.21,0.0,0,0,0,0, 0,0,.81,.50,.14,
0.0,0,0,0,0,0,0,0.0,0,0,0,0,.68,.28,0.0,0.0,0.0, 0.0,0,0,.13,0,0,0,0,0,0,0,.16

Appendix D

D.1 Regular expression notation of PERL

`.`	Matches any character except newline
`[a-z0-9]`	Matches any single character of set
`[^a-z0-9]`	Matches any single character *not* in set
`\d`	Matches a digit, same as [0–9]
`\D`	Matches a non-digit, same as [^0–9]
`\w`	Matches an alphanumeric (word) character [a–zA–Z0–9_]
`\W`	Matches a non-word character [^a–zA–Z0–9_]
`\s`	Matches a whitespace char (space, tab, newline...)
`\S`	Matches a non-whitespace character
`\n`	Matches a newline
`\r`	Matches a return
`\t`	Matches a tab
`\f`	Matches a formfeed
`\b`	Matches a backspace (inside [ ] only)
`\0`	Matches a null character
`\000`	Also matches a null character because...
`\`*nnn*	Matches an ASCII character of that octal value
`\x`*nn*	Matches an ASCII character of that hexadecimal value
`\c`*Y*	Matches an ASCII control character
`\`*metachar*	Matches the character itself (|, \., *...)
`(abc)`	Remembers the match for later backreferences
`\1`	Matches whatever first of parens matched
`\2`	Matches whatever second set of parens matched
`\3`	and so on...
x`?`	Matches 0 or 1 *x*'s, where *x* is any of above
x`*`	Matches 0 or more *x*'s
x`+`	Matches 1 or more *x*'s
x`{`*m*`,`*n*`}`	Matches at least *m* *x*'s but no more than *n*
`abc`	Matches all of a, b, and c in order
`fee|fie|foe`	Matches one of fee, fie, or foe
`\b`	Matches a word boundary (outside [ ] only)
`\B`	Matches a non-word boundary
`^`	Anchors match to the beginning of a line or string
`$`	Anchors match to the end of a line or string

Appendix E

E.1 Roman Coding Scheme for Devnagri used in the thesis

a	A	i	I	u	U	q
अ	आ	इ	ई	उ	ऊ	ऋ
e	E	o	O	M	H	z
ए	ऐ	ओ	औ	ं	ः	ँ
k	K	g	G	f		
क	ख	ग	घ	ङ		
c	C	j	J	F		
च	छ	ज	झ	ञ		
t	T	d	D	N		
ट	ठ	ड	ढ	ण		
t	T	d	D	n		
त	थ	द	ध	न		
p	P	b	B	m		
प	फ	ब	भ	म		
y	r	l	v	S		
य	र	ल	व	श		
S	s	h	kR			
ष	स	ह	क्ष			

XAMPLE : hiMdI akSaramAlA = हिन्दी अक्षरमाला

Appendix F

F.1 Roman Coding Scheme for Devnagri used in the computer system

a	A	I	I	u	U	q	e	E	o	O	M	H	.z
अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ	ं	ः	ँ
				k	K	g	G	f					
				क्	ख्	ग्	घ्	ङ्					
				c	C	J	J	F					
				च्	छ्	ज्	झ्	ञ					
				t	T	d	D	N					
				ट्	ठ्	ड्	ढ्	ण्					
				w	W	∝	X	n					
				त्	थ्	द्	ध्	न्					
				p	P	b	B	m					
				प्	फ्	ब्	भ्	म्					
				y	r	l	v						
				य्	र्	ल्	व्						
				S	R	s	h						
				श्	ष्	स्	ह्						

Examples:

rAma	kqRNa	JFAna	Sawru	AzKa	yakRa
राम	कृष्ण	ज्ञान	शत्रु	आँख	यक्ष

Bibliography

[1] Akshar Bharati and Sangal Rajeev, August.(1990), *A Karaka Based Approach to Parsing of Indian Languages*, In COLING90: Proc. of Intl. Conf. on Computational Linguistics, Helsinki, Association for Computational Linguistics, NY, August 1990..

[2] Akshar Bharati,Chaitanya Vineet, Patnaik,B.N. and Sangal, Rajeev, Mar.(1991), *Meaning and Natural Language Processing* TRCS-91-116,Dept. Of CSE, IIT Kanpur.

[3] Akshar Bharati,Chaitanya Vineet and Sangal Rajeev,Feb.(1990), *A Computational Grammar for Indian Language Procesing*, TRCS-90-96,Dept. of CSE, IIT,Kanpur.

[4] Akshar Bharati,Chaitanya Vineet and Sangal Rajeev, (1995), *Natural Language Processing : A Paninian Perspective* , Prentice Hall of India Pvt. Ltd.., 1995

[5] Akshar Bharati,Chaitanya Vineet and Sangal Rajeev,(1994), *Paninian Framework and its Application to Anusaraka*, Sadhna,Vol.19,Part 1,February 1994, pp.113-127.

[6] Akshar Bharati,Bhargava Y Krishna, and Sangal Rajeev (1993), *Reference and Ellipsis in an Indian Languages Interface to Databases*, Computer Science and Informatics Journal 23, 3, Sept. 1993, pp. 60-82.

[7] Boon Low Hwee, (1992), *Some Lessons Learnt by a New Comer.*, MT Summit IV., pp. 181-182.

[8] Brent Michael.R, (1991), *Automatic Acquisition of Subcategorization Frames from untagged text*, 29th Proceedings ACL-91,pp.209-214

[9] Brown P.F,Cocke.J,et.al. (1990), *A Statistical Approach to Machine Translation.*, Computational Linguistics., Vol.16,No.2,1990

[10] Charniak Eugene, (1993), *Statistical Language Learning* , MIT Press, 1993

[11] Doi Shinichi and Muraki Kazunori, (1993), *Evaluation of DMAX Criteria for Selecting Equivalent Translation based on Dual Corpora Statistics*, TMI-93.,pp.302-311.

[12] Hindle Donald and Rooth Mats,(1991), *Structural Ambiguity and Lexical Relations*, 29th Proceedings, ACL-91,pp.229-236.

[13] Manning Christopher D., (1993), *Automatic Acquisition of a Large Subcategorization Dictionary from Corpora*, 31st Proceedings ACL-93, pp. 235-242.

[14] Ravishankar P.V., (1992), *Enhancements to the Linguists' Workbench*, M.Tech thesis ,MT-CS-92-20,Dept. of CSE, IIT,Kanpur.

[15] Rohrer Christian,(1992), *The Future of MT Technology* , MT Summit IV.,pp. 195-196.

[16] Sadler V.,Vendelmann R.,(1990) *Pilot Implementation of Bilingual Knowledge Bank* , COLING-90,Vol.3,pp. 449-451

[17] Sato S.,Nagao M.,(1990) *Towards Memory Based Translation.* , COLING-90,Vol.3,pp. 247-252

[18] Schabes Y.,(1992) *Stochastic Lexicalized Tree-Adjoining Grammars.*, COLING-92,pp. 425-432

[19] Srinivas B., (1991), *Linguistics Workbench- A Grammar Development tool for Indian Languages*, M.Tech thesis,MT-CS-91-15 I.I.T, Kanpur

[20] Sumita Eiichiro, and Iida Hitoshi, (1992), *Example-based NLP Techniques- A case study of Machine Translation* AAAI-92 Workshop "Statistically Based NLP Techniques", AAAI Technical Report W-92-01,1992